Patent Application

Memory Management System and Algorithm for Network Processor Architecture

Inventor(s): Ryszard Bleszynski, residing at
18817 Bellgrove Circle
Saratoga, CA 95070
A citizen of Indonesia

Man Dieu Trinh, residing at
2215 Emerald Hills Drive
San Jose, CA 95131
A citizen of the United States

Company: Bay Microsystems, Inc.
2700 Augustine Dr., Suite 298
Santa Clara, CA 95054

Background of the Invention

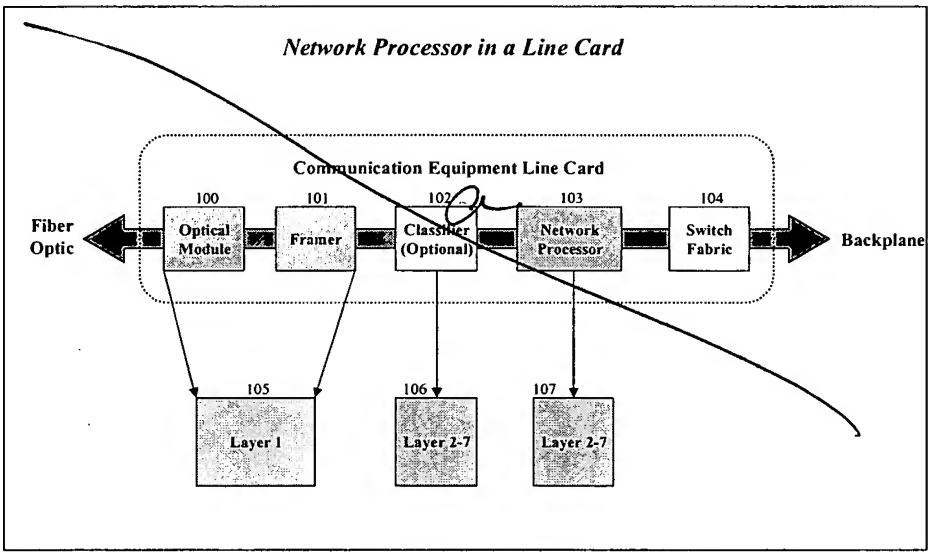
a FIELD OF THE INVENTION

The present invention relates in general to the elimination of packet and control memory bottlenecks in communication networks and more specifically to a Network Processor (NP) architecture. The architecture and algorithm incorporated in the Network Processor Integrated Circuit (IC) of the invention provide orders of magnitude faster performance and the scalability needed to meet the explosively-increasing demand for bandwidth.

a BACKGROUND INFORMATION

Almost all communications equipment uses one or more network processors. Communications equipment includes but is not limited to: high-speed routers, switches, intelligent optical devices, DLSAM, broadband access devices, voice gateways, etc. The equipment may deploy the NP in a centralized or distributed manner. Distributed NP is popular for high speed and intelligent communications equipment. For lower and mid-range equipment, centralized NP is very attractive since this will keep the equipment price very low. In complex, high-speed intelligent broadband equipment, the NPs (such as those manufactured by Intel or Lucent) are distributed and each line card may contain one or more NPs. Figure 1 illustrates the NP physical location within the line card and logical functions within the Networking stacks.

a ~~Figure 1~~

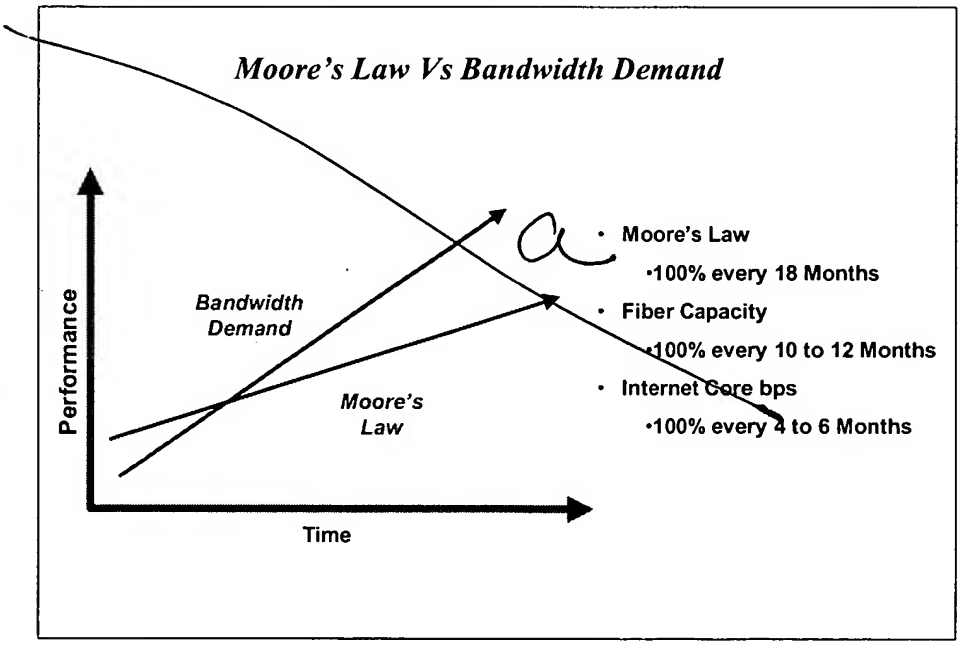


In a typical line card, the fiber-optic cable is connected to the optical module. The other end of the fiber optic line typically connects to an external router or another communications device. Among other functions, the optical module converts the optical signal into an electrical signal. The optical module presents the electrical signal to the framer. The framer performs functions such as: framing, error checking and statistical

gathering. The framer provides the framed information to the optional classifier. The classifier performs a flow classification function. The classifier is an optional function. Most equipment does not require classification beyond layer three or four and most network processors perform at least up to layer three or four. The network processor processes the information and forwards it into the appropriate line card within the system's backplane using the switch fabric. Logically, the optical module and framer perform layer one of the OSI stack, whereas the NP and optional classifier handles layers 2 through 7. Processing intelligence, power and bandwidth capacity are the biggest differentiation factors between Network Processors.

Among the single biggest limiting factor for NPs to scale and meet increasing Internet bandwidth demand is Moore's law. Moore's law limits the advancement in semiconductor process technology to 18 months in order to achieve a 100% performance improvement. Doubling every 18 months is far below the Internet bandwidth demand, which doubles every four to six months. As of today, early generation network processors cannot scale by 4 or 16 within a two to three year time window. Overcoming Moore's law is a non-trivial process. Figure 2 illustrates Moore's law versus Bandwidth demand curve.

~~Figure 2~~



The current techniques in network processor architectures are bounded by Moore's law. In general there are three approaches to NP architectures: Multiple RISC, Configurable hardware and a mixture of RISC and hardware. The RISC Architecture and Instruction Set was created decades ago for devices geared toward human to machine interaction. Network devices are not human to machine devices. They are machine-to-machine devices. In other words, they are communicating to high-speed machines and not to humans. Multiple RISC engines within the data path of networking equipment will not meet the required bandwidth demand. Moore's law is one limiting factor. Another severe

limiting factor is the complexity of the software compiler, scheduler and/or kernel to efficiently control and maximize the processor's operation. Creating a mini operating system is not the solution to the explosive demand in bandwidth, especially when Moore's law (hardware) cannot even meet the demand.

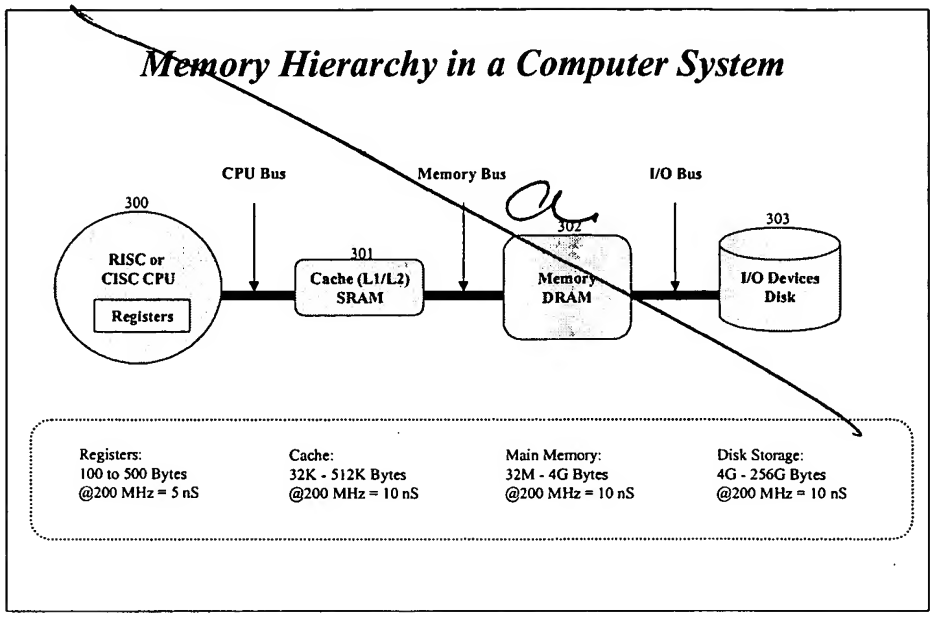
Configurable hardware results in the highest-performance processors. The simple software interface avoids any performance degradation. Eliminating any software within the information path and replacing it with configurable gates and transistors significantly boosts the performance of the Network Processor. At the gate level, without any creativity within the architecture, Moore's law still bounds the performance advancement of Network Processor architecture.

A mixture of multiple RISCs and configurable hardware machines has two different flavors. The first flavor uses the RISCs in the data path and the other one is to have the RISC processor in the control path. Traditionally, RISC processors in the control path have been limited to those external to the NP.

In addition to the processing capability of the Network Processor, another critical bottleneck in the Network Processor architecture is the memory throughput for the payload buffer. Memory technology advancement is also bounded by Moore's law. Today's generation of store and forward network processors use a single hierarchy memory organization. Bandwidth may be increased by increasing the width of the memory bus. Increasing the information width of the packet memory bus, however, only decreases the actual memory throughput for packet sizes smaller than the bus width because of the additional processing overhead.

Figure 3 illustrates a typical memory hierarchy within a computer system using either a RISC or CISC CPU.

Figure 3



Due to the principle of locality, the linear multilevel memory hierarchical scheme of Figure 3 works very well in a CPU architecture. The CPU contains very high speed registers for immediate access. These registers are high-speed memory internal to the CPU providing the CPU with very high-speed single cycle access to the information. The cache is a small piece of memory and has a slightly slower access time compared to the registers. As the memory moves away from the CPU, the storage capacity increases and the access time decreases.

Caching theory works well in the computer architecture, but, unfortunately, due to the non-deterministic nature of network traffic, caching does not work well for Network Processors. The principle of locality does not apply in networking.

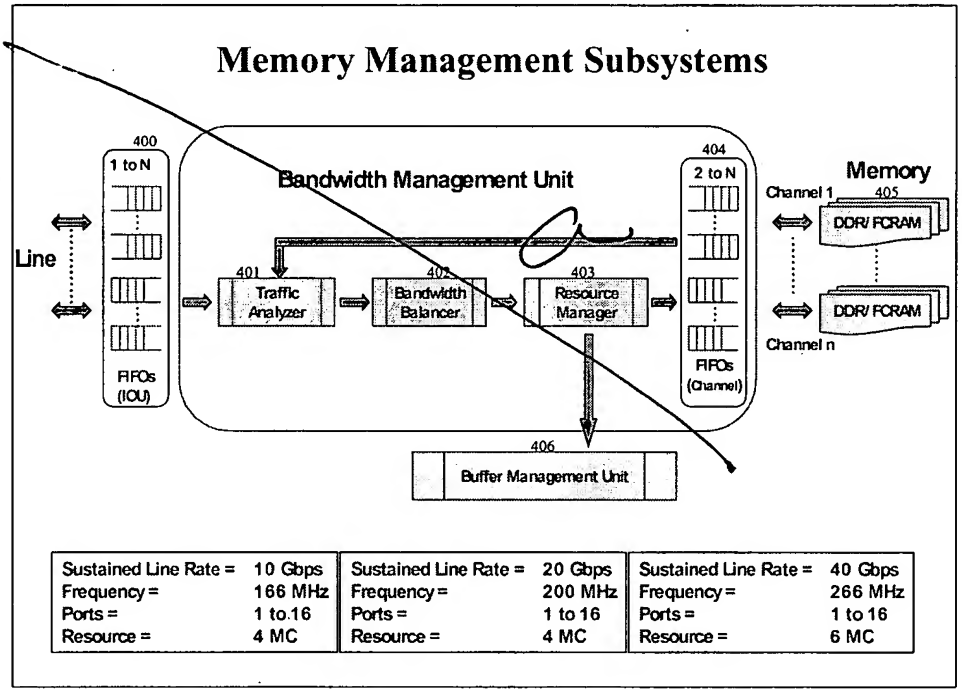
INS A1

Summary of the Invention

The present invention provides novel techniques for balancing memory accesses to sustain and guarantee a desired internet bandwidth (line rate) demand under any traffic pattern using low cost DRAM. Among other reasons, Network Processor advancements in meeting the explosively-increasing demand requirements for Internet bandwidth cannot rely on traditional memory locality principles. In particular, the technique of the present invention provides a novel traffic analyzer and memory bandwidth-balancing algorithm that will maximize aggregate memory bandwidth using low cost commercial DRAM memory, and enable true scalability for NPs and advancement independent of improvements in memory capabilities and Moore's law.

According to the invention, systems and methods are provided for maximizing the memory throughput by dividing the memory into sub-channels. The memory hierarchy is a single level as opposed to the linear multilevel approach used in computer systems. Each channel may consist of single or multiple banks of DRAM and have a 64-bit wide information path. The algorithm can be applied to channel granularity other than 64-bits. Due to the long latency of low cost DRAM, it can be mathematically proven that four individual 64-bit wide memory channels provide significantly better performance than a single 256-bit wide memory, especially for smaller packet sizes. Figure 4 illustrates a top-level block diagram for the Memory Management Subsystems of the present invention.

Figure 4



The bandwidth management unit in Figure 4 resides within the NP. The framer or classifier not shown in the figure is located between the fiber optic line and the NP. Each fiber line connects to one or more external routers or another communications device. As each packet or cell arrives, it is temporarily stored within the ingress FIFOs of Input Output Unit (IOU) 400. A Bandwidth Management Unit (BMU) consists of a traffic analyzer 401, a bandwidth balancer algorithm 402, a resource manager 403 and payload channel FIFOs 404. (To clarify the figure, please note that packets do not actually pass through the traffic analyzer, the bandwidth balancer or the resource manager.) The traffic analyzer analyzes the traffic by using counters to measure the depth level of the payload channel FIFOs 404. The count values are used by the bandwidth balancer to apply the balancing algorithm. The bandwidth balancer algorithm balances the traffic load across the multiple channels. The resource manager interfaces with the buffer management unit 406 for pointer allocation and recycling.

The payload channel FIFOs on the memory side provide additional temporary storage to compensate for latencies inherent within the DRAM memory systems. As illustrated in Figure 4, the memory channels may consists of two or more channels. Each channel bus width used in this example is 64-bits wide. The algorithm may also, for example, be applied to channel bus widths of 2 to n where n is a positive integer.

In a store and forward architecture, network traffic arrives from the line side and the NP temporarily buffers the information in the memory side. This buffering is necessary to provide the tolerance needed against network congestion and to process traffic engineering and forwarding functions to determine the next hop destination of the packet data. After the network processor determines the destination, the traffic leaves the processor from the memory into the line side. In this example the buffer granularity is 64-bytes. The present invention can be applied to buffer size other than 64-bytes.

The present invention guarantees and sustains a line rate of, for example, 10 Gbps using four memory channels at a memory bus frequency of 166 MHz. Increasing the frequency to 200 MHz, the algorithm guarantees and sustains up to 20 Gbps of line rate. With 6 channels at 266 MHz, 40 Gbps of usable memory bandwidth is achievable. These numbers apply to packet sizes of 40-bytes or greater.

For the example here, four channels and a 64-byte buffer size are used. In an ideal scenario and the simplest case, when a packet or cell arrives, each 64-byte chunk is stored in one memory channel in a sequential manner. In particular, the first 64-byte goes to channel one, and the next one goes to channel two and so forth. This simplest case works fine if the outgoing (egress) traffic pattern is deterministic. Due to the non-deterministic outgoing traffic pattern experienced in real-world networks, however, the memory channels may not be balanced and thus the aggregate memory bandwidth will fall below the line rate. With unbalanced memory accesses, one or two channel swamp may occur and the line rate cannot be sustained or guaranteed. This is true since each channel provides significantly less bandwidth than the line rate requirement. In other words, because different packet streams are read out at different rates due to the demands of different-quality services (e.g., DSL, T1, etc.), these varying demands will affect each channel differently and generally cause the rate of reading an information unit from one network memory channel to be different from the reading rates of the other memory channels. The non-deterministic nature of the reading rate makes it particularly important to the determination of traffic flow.

According to an aspect of the invention, a method is provided to analyze the incoming and outgoing traffic patterns. The method, a traffic analyzer, analyzes incoming and outgoing traffic by monitoring the depth level of the FIFOs. In one embodiment, the traffic analyzer uses counters to measure the FIFO depth level.

In another aspect of the invention, an algorithm is provided to intelligently determine the channel selection for storing the incoming traffic. The algorithm balances the channels appropriately depending on the incoming and outgoing traffic patterns. The algorithm allocates a memory line by fetching the corresponding pointers for the line (which, for example, may consist of four 64-byte buffers) from the buffer management unit. A memory line may consist of two to N buffers, each of which may be assigned to a

channel. In the example, the algorithm fetches four pointers, one for each buffer. Under severe traffic patterns, the algorithm includes intelligence to sacrifice one or more 64-byte buffers without even using the buffer space in exchange for a new line. The new line consists of four 64-byte buffers and thus increases the channel selection choice.

INS A2

Description of the Specific Embodiments

The present invention provides a system and method for controlling accesses to a memory, wherein the memory is divided into two or more memory channels. The memory is addressed line by line. Each memory line is divided into buffers, each of which may be assigned to a corresponding channel. (In some instances, a buffer is not assigned to a channel if it is sacrificed, as described below.) A buffer is considered to be assigned to a channel when it is permitted to store data in that memory channel. Each memory line is pointed to by a line pointer, and each buffer is pointed to by a channel pointer. Using a bandwidth balancing algorithm of the invention, the invention selects into which memory channel an information unit is to be stored. The information unit may be, for example, a complete packet if the size of the buffer can accommodate the entire packet, or a portion of a packet, if the buffer is only large enough to accommodate a portion of the packet. The information unit may also be, for example, an ATM cell. The memory may be DRAM for storing data in a network processor. Each memory channel may comprise one or more banks of DRAM. While the data is stored in the memory, the network processor or other communications device determines, among other things, to which destination (e.g., a next hop router) the data should be sent.

The bandwidth balancer algorithm determines channel selection based upon load parameters, such as, for example, parameters relating to incoming and outgoing traffic for a channel, or, more particularly, the number of currently pending read and write requests and the active buffer count for each one of the four channels. In one embodiment (with no buffer sacrifice option), the BBA (Bandwidth Balancer Algorithm) will select the channel for the incoming cell, packet or packet segment using the following order of criteria:

1. Of the unoccupied channels in a line of memory, the channel with the lowest number of read requests is selected because a read request has the highest priority.
2. If there is more than one available channel within the line with the lowest number of read requests, then, of those channels, the channel with the lowest number of write requests is selected.
3. If the number of write requests pending for those channels is the same, then of those channels, the channel with smallest number of active buffers is selected. (A buffer is active if it stores data to be read out.) This criterion is based on a statistical prediction related to the fact that the channel with the higher number of active buffers will eventually generate more reads from that channel.
4. If all the channels determined by 1, 2 and 3 above have the same number of active buffers, then channel with a smallest ID is chosen.

In a store and forward architecture, such as that of the present invention, the information stored in the memory will eventually be read out for forwarding or discarding. Based on this fact, the method in the present invention uses counters for each channel to keep track of the number of buffers pending to be read, pending to be written, and active. Each channel uses three counters, pending read request (PRR), pending write request (PWR) and active buffer (AB) count as illustrated in Figure 5.

In Figure 5, the Policy Control Unit (PCU) 512 resides within the NP. Among other functions, the PCU performs functions such as usage parameter control (UPC) on information units (e.g., packets) arriving on the ingress line FIFOs 517. When the PCU completes its operations on an incoming cell or packet segment, the PCU initiates a write request to the Data Buffer Unit (DBU) 514 to write data into the memory 508. As explained in detail below, the BBA within the DBU selects the memory channel into which the information unit is to be stored. Each channel in memory 508 corresponds to a write request (control) FIFO 519 and associated incoming payload (data) channel FIFO 511. The PCU transmits the write request to the DBU through the request FIFO 519 corresponding to the selected memory channel, and temporarily stores the information unit in the incoming channel FIFO 511 corresponding to the selected memory channel before the information unit is written into the selected memory channel. The reason request and payload channel FIFOs are used to store requests and data, respectively, is that accesses to DRAM are subject to nondeterministic latencies.

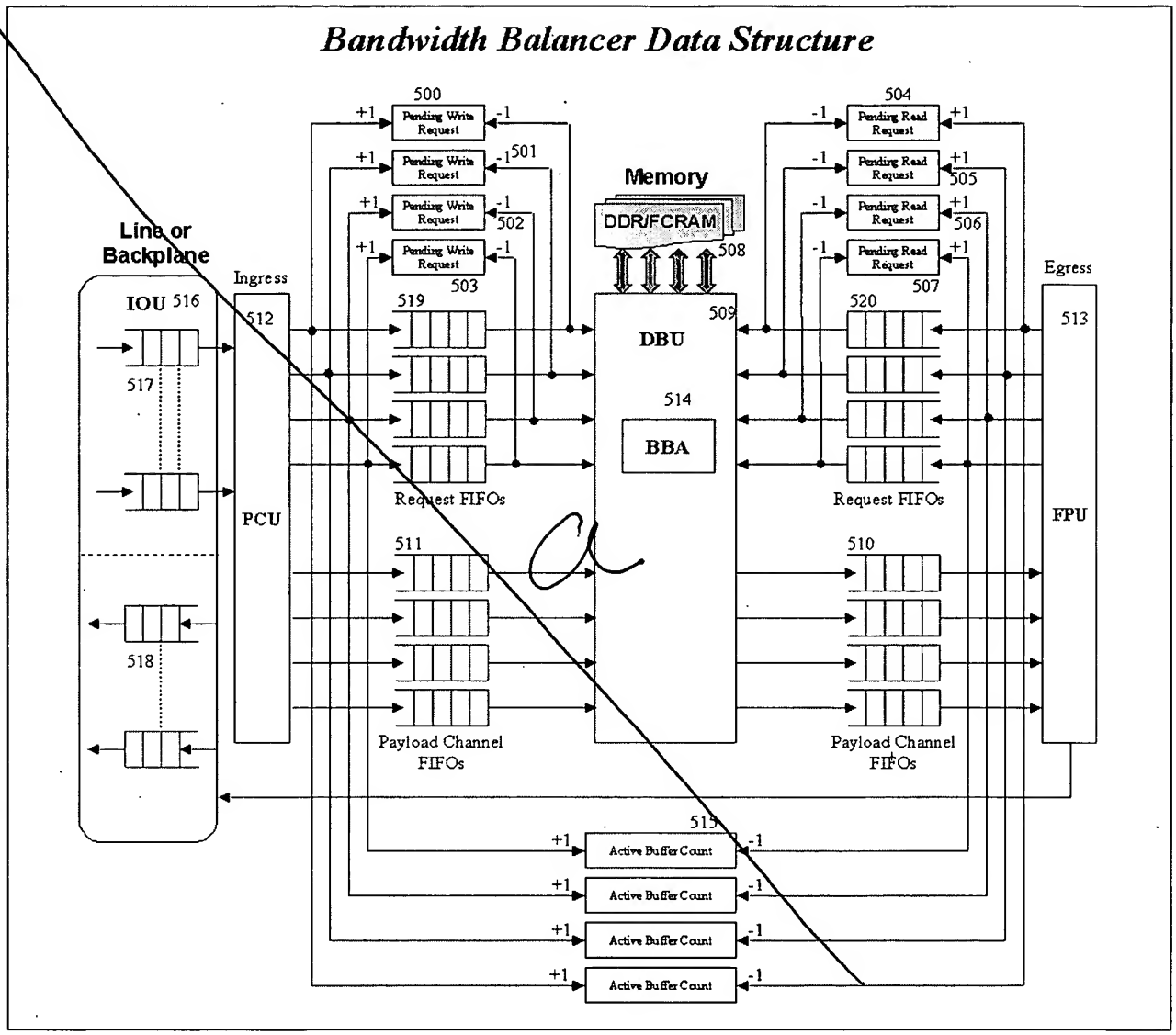
Among other actions, the Forwarding Processing Unit (FPU) 513 performs a forwarding function, including calculation of the next destination (e.g., next hop). When the FPU completes its operations for an outgoing cell or packet segment, the FPU initiates a read request to the DBU to read data from the memory 508. Each channel in memory 508 corresponds to a read request (control) FIFO 520 and an associated outgoing payload (data) channel FIFO 510. The FPU transmits the read request to the DBU through the read request FIFO 520 corresponding to the channel storing the requested information unit. After the DBU reads the information unit from a buffer in memory, the information unit is stored in an outgoing payload channel FIFO 510 corresponding to the channel from which the information unit came. The information unit is then transferred to an egress line FIFO 518, after which the information unit is forwarded to its next destination.

When the PCU generates a write request the method increments the pending write request counter value by one. When the DBU services the write request (moves data from the incoming payload channel FIFO to a buffer in memory), the method decrements the pending write request counter (500, 501, 502 or 503) value by one. When the FPU generates a read request the method increments the pending read request counter (504, 505, 506 or 507) value by one. When the DBU services the read request (moves data from a buffer in memory to an outgoing payload channel FIFO), the method decrements the pending read request counter value by one. When the PCU generates a write request the method increments the active buffer counter 515 value by one. When the FPU generates a read request the method decrements the active buffer counter 515 value by one.

One of ordinary skill in the art will recognize that the present invention is not limited to using the exemplary counters described herein, but can use any technique to measure the load on the memory channels.

Figure 5 illustrates the counters used for the method in this invention.

~~Figure 5~~



For four channels, the method uses 12 counters. One memory line consists of four 64-byte buffers. When the resource manager fetches a pointer, the buffer management unit provides a line pointer. The resource manager fetches a new line pointer when the BBA requires a new line to be fetched, as indicated in Figure 7.

The line pointer points to a memory line which consists of four buffers, each of which may be assigned to a channel. Each time a 64-byte quantity of information is ready to be stored, the BBA selects from among the four channels. The PCU maintains state information using a field called the Payload Channel Occupancy (PCO) to identify which of the four channels are occupied. For example, if the buffers in a line corresponding to channels 1 and 3 are occupied, the PCO vector for that line would be (1,0,1,0) where the element vectors correspond to channels (3,2,1,0) in that order. We refer to a channel as being "occupied" or "unavailable" with respect to a particular memory line if, within that line, the buffer that corresponds to the channel stores data. We refer to a channel as being "written" with respect to a particular memory line to describe the writing of data within that line into a buffer corresponding to the channel. The relationship between the buffers and the channels is maintained in the Channel Sequence Table, as explained below. The PCO is a four-bit field for each memory line that is maintained in a separate structure called the Policy Control State (PCS) within the PCU.

Initially, when the resource manager fetches a new line from the Buffer Management Unit (Figure 4), the BBA can select any one of the four channels. The corresponding bit in the PCO field is set to logic one to indicate when a particular channel is already occupied. When the next cell or packet segment arrives after the first channel is selected to be written by the BBA, the BBA can select any one of the remaining unoccupied three channels. When only one or two channels left, the selection is constrained to one or two channels.

When the PCO state indicates that there are only one or two channels left and the channel selection does not meet any of the above four balancing criteria, the BBA includes an option to sacrifice one or more buffers (e.g., 64-bytes of the 256 Mbytes of memory) for performance trade off. Under appropriate load conditions, when the channel sacrifice (skipping) option is enabled the resource manager fetches a new memory line pointer (i.e., allocates a new line) and this provides the BBA with four new buffers, one buffer per channel, to choose from instead of one or two. The algorithm can be applied when one or two channels are available from the selection and they do not meet the channel selection criteria. One or two buffers can be sacrificed for performance. In other words, if instead the algorithm stored the data in one of the two remaining buffers, this would load the corresponding channel that stores the data beyond a limit deemed acceptable according to the four balancing criteria.

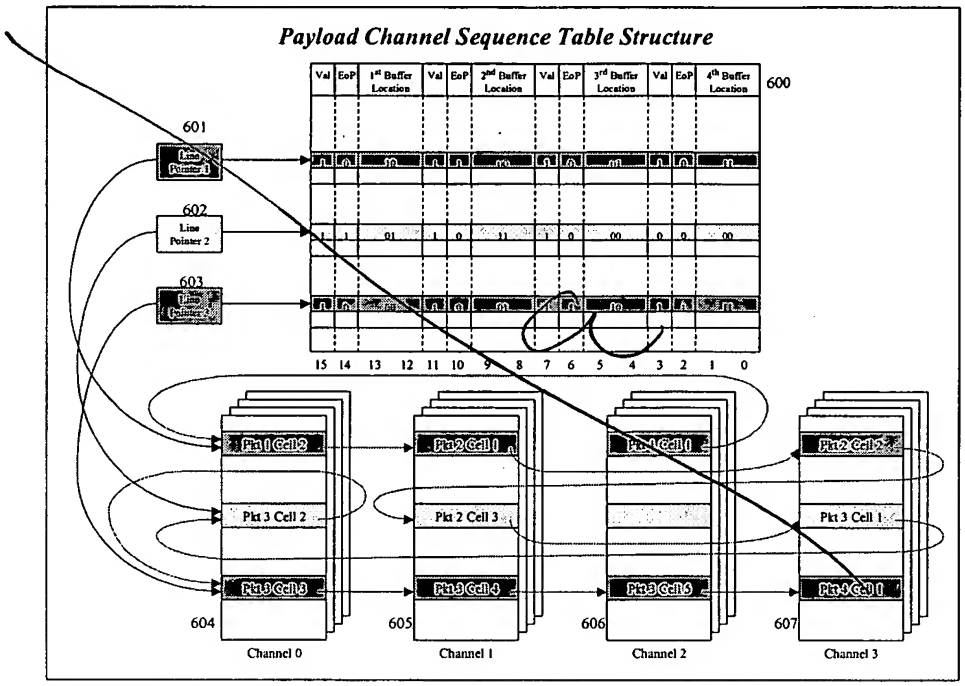
Buffer Link List

In the architecture of the present invention, the next buffer pointer is stored in the header of the current buffer. The next buffer pointer is written in the header section of the buffer at the same time as the payload using a burst-write transaction. In order for the present invention to enable dynamic channel selection, the payload channel occupancy state information within one memory line cannot reside within the payload buffer header.

The sequence of every four cells or packet segments is dynamic and it is determined by the BBA. The payload channel occupancy state information is kept in a separate data structure.

The method in the present invention uses a separate data structure to maintain the sequence of channel usage. This data structure, called the Channel Sequence Table, is illustrated in the figure 6. The CST may be stored in SRAM or embedded DRAM.

Figure 6



The CST illustrated in Figure 6 contains information about the sequence of the channel occupancy within the memory line. In this example, a memory line consists of four 64-byte buffers one buffer from each channel. Since one packet may occupy one or more buffers, the buffer sequence within a packet must be maintained. Initially in this example, buffer one contains the first segment of the packet (packet 1, cell 1), buffer two contains the second segment (packet 1, cell 2) and so forth.

The first buffer location field within the payload channel sequence table in Figure 6 contains the channel number (represented in binary) to which the first buffer is assigned. The second buffer location contains the channel number where the second buffer resides. The third buffer location contains the channel number where the third buffer resides. Since there will be some occasions that one or two of the buffers within a line are not used because they fail to meet the balancing criteria (and are thus sacrificed), the method uses a valid bit within the CST data structure to indicate whether the buffer is occupied. As illustrated in the second line of the CST, a valid bit of zero for the fourth buffer location indicates that that buffer is being sacrificed.

According to the present invention, the CST serves two purposes. It provides real-time dynamic channel assignment for the BBA. In addition, the CST enables a pre-fetch method for the FPU in the UBR (Unassigned bit rate) or packet mode of operation. In UBR and packet mode, the FPU forwards cells and packets one packet at a time. In conjunction with the first segment of the packet, the FPU can fetch one entry from the

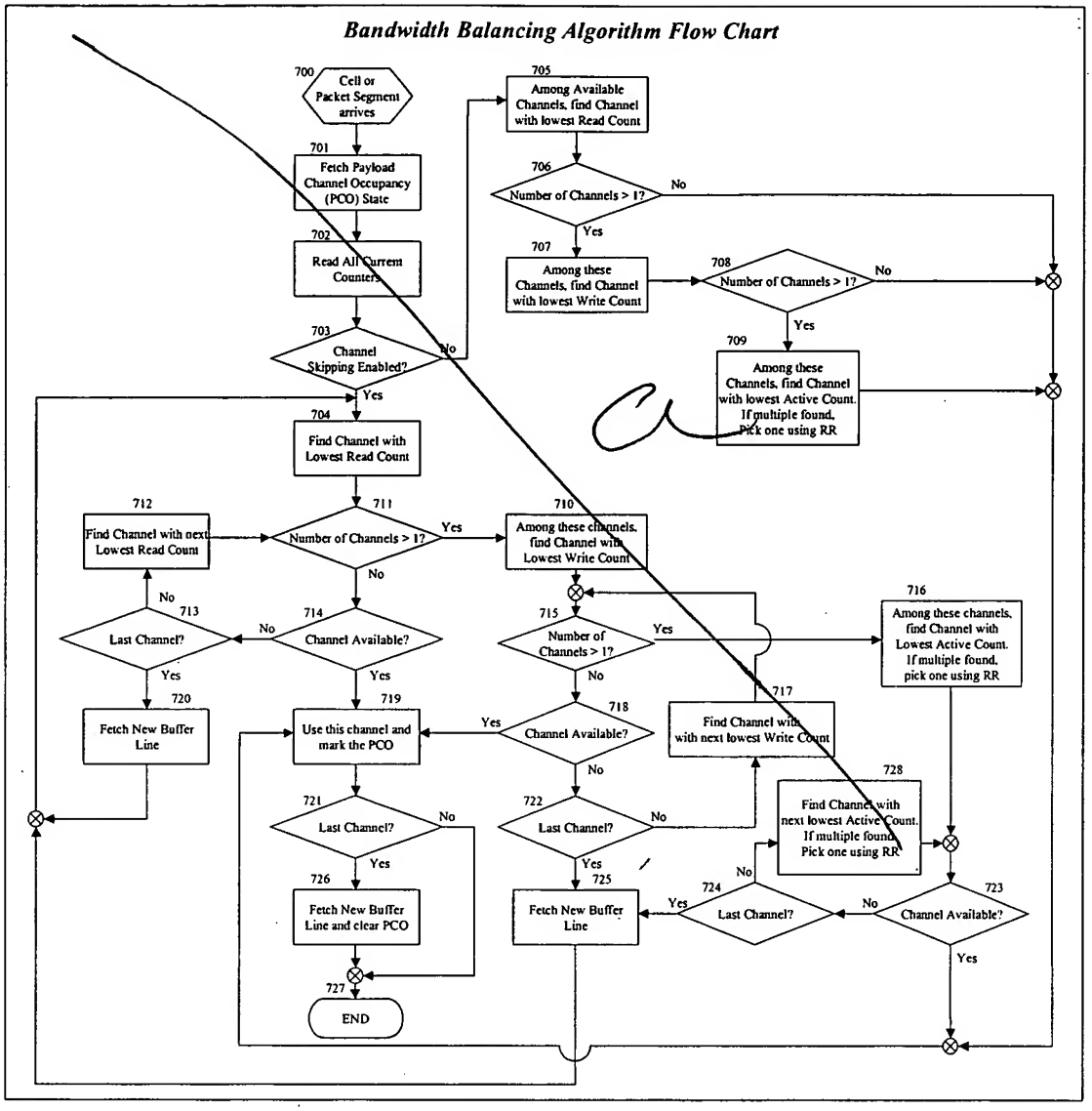
Channel Sequence table and know exactly the sequence of channels in which to send the read requests to the DBU to fetch the information from memory in advance. The pre-fetch method in the FPU provides a tremendous increase in throughput especially for large packets spanning more than one memory line or four buffers in this particular example.

The method also uses additional information within the CST, the EOP (End of Packet) field. The PCU sets the value of the EOP bit to one to mark the end of packet. This information allows the FPU to pre-fetch the sequence information until it encounters the buffer with the EOP field set to one.

In the present implementation of this invention, the CST structure resides in a separate memory region. The memory used in this region is the ZBT SRAM, which provides data every cycle. The bandwidth requirement on this interface is 2 reads and 1 write. The PCU does a read-modify-write to update the CST table. The FPU only reads and uses the information contained within the table.

The complete dynamic channel selection process is given by the algorithm illustrated by the flow chart in figure 7

~~Figure 7~~



After the information unit (e.g., cell or packet segment) arrives (step 700), the PCU provides the PCO state information to the BBA (step 701) and the BBA reads the values of all the current counters (step 702). The first test the BBA performs is whether the channel sacrifice (skipping) option is enabled (step 703). The option may be enabled by the programmer.

Channel Skipping Not Enabled

If this option is not enabled, the BBA will determine from among the available (unoccupied) channels the channel with the lowest number of pending read requests (step 705).

If only one channel has a lowest read count (step 706), then the BBA will select this channel for storage of the information unit (e.g., packet segment), and indicate in the

PCO that the channel is occupied by setting to a logic one an indicator in the appropriate field of the PCO corresponding to the occupied buffer in the memory line (step 719). If the selected channel is not the last channel within the line (step 721), then the algorithm for this information unit ends (step 727), and the BBA will wait for the next information unit to arrive. If the selected channel is the last channel (step 721), then the BBA will fetch a new line pointer (thereby allocating a new line in memory) and initialize the PCO to zero for that line before exiting the algorithm (step 726).

If more than one channel has a lowest read count (step 706), then, from among those channels, the BBA determines which channel has the lowest number of pending write requests (step 707). If only one channel has both the lowest read and write counts (step 708), then the BBA selects this channel for storage of the information unit, and proceeds with the step of marking the PCO (step 719) followed by the other steps implemented above in the case of only one lowest-read count channel (i.e, the last channel test).

If, however, more than one channel has both the lowest read and write counts (step 708), then from among those channels, the BBA determines the channel that has the lowest number of active buffers. If more than one channel matches all three criteria, then the BBA uses a round robin selection from among those channels based upon an ascending or descending order of the channel identification numbers, which are arbitrarily assigned as is well known in the art (step 709). The BBA selects the channel that survives these tests, and marks the PCO accordingly (step 719).

Channel Skipping Enabled

If the channel skipping option is enabled, the BBA will determine the channel with the lowest number of pending read requests (step 704).

If only one channel has a lowest read count (step 711), then the BBA checks using the state information in the PCO whether the channel is available (unoccupied) (step 714). If the channel is available, the BBA selects the channel for storage of the information unit, marks the PCO (step 719) and performs the last channel test (steps 721 and 726, if necessary).

If, however, the lowest-read-count channel is not available (step 714), then the BBA performs a last channel test and fetches a new line if the channel is the last channel (steps 713 and 720). Then the BBA starts again at the first determination of the channel with the lowest read count (step 704) to ultimately determine in which channel of the new line the data should be stored.

If the lowest-read-count channel is not available (step 714) and not the last channel (step 713), the BBA finds the channel with the next lowest read count value (step 712). If (1) there is only one channel with this next lowest-read-count value (step 711), and (2) it is available (step 714), then the BBA selects this channel for storage, marks the PCO accordingly (step 719), and performs the last channel test (steps 721 and 726, if necessary).

If more than one channel has the lowest read count (step 711), then, from among those channels, the BBA determines which channel has the lowest number of pending write requests (step 710). If only one channel has both the lowest read and write counts (step 715), then the BBA determines whether this channel is available (step 718). If it is, then the BBA selects this channel for storage of the information unit, marks the PCO (step 719) and performs the last channel test (steps 721 and 726, if necessary).

If the channel is not available (step 718), then the BBA determines whether that channel is the last channel capable of being assigned in the line (step 722). If it is, then the BBA fetches a new buffer line (step 725). Then the BBA starts again at the first step of determining the channel with the lowest read count (step 704) to ultimately determine in which channel of the new line the data should be stored.

If the channel is not the last channel (step 722), then the BBA finds the channel having both the lowest read count and the next lowest write count (step 717). The BBA then again makes the determination whether there is more than one channel meeting these criteria (step 715), going through the loop again.

If, however, more than one channel has both the lowest read and write counts (step 715), then from among those channels, the BBA determines the channel that has the lowest number of active buffers, or, if more than one channel matches all three criteria, then the BBA uses a round robin selection from among those channels based upon an ascending or descending order of the channel identification numbers (step 716).

The BBA then determines whether the channel that survives all these tests is available (step 723). If it is, then the BBA selects it for storage, marks the PCO (step 719) and performs the last channel test (steps 721 and 726, if necessary). If, however, the channel is not available (step 723), then the BBA determines whether the channel is the last channel in the line (step 724). If it is not the last channel, then the BBA determines the channel having both the lowest read and write counts as well as the next lowest active buffer count, or, if more than one channel matches all three criteria, then the BBA uses a round robin selection from among those channels based upon an ascending or descending order of the channel identification numbers (step 728). The BBA then performs the channel available test again (step 723).

If the channel is the last channel (step 724) then the BBA fetches a new buffer line (step 725). Then the BBA starts again at the first step of determining the channel with the lowest read count (step 704) to ultimately determine in which channel of the new line the data should be stored.

The selection algorithm represented by the above flowchart is only one example of the implementation of the BBA, and should not be viewed as limiting the scope of the invention. The invention can, for example, employ other algorithms using other count mechanisms with a similar or different sequence of tests in order to allocate incoming information units among memory channels.